

Министерство образования и науки Российской Федерации
Московский физико-технический институт (государственный университет)

Физтех-школа радиотехники и компьютерных технологий
Кафедра системного программирования ИСП РАН
Лаборатория (laboratory name)

Выпускная квалификационная работа бакалавра

Исследование и разработка методов машинного обучения

Автор:

Студент 082 группы
Иванов Иван Иванович

Научный руководитель:

научная степень

Денисов Денис Денисович

Научный консультант:

научная степень

Сергеев Сергей Сергеевич



Москва 2025

Аннотация

Исследование и разработка методов машинного обучения
Иванов Иван Иванович

Краткое описание задачи и основных результатов, мотивирующее
прочитать весь текст.

Abstract

Research and development of machine learning methods

Содержание

1	Введение	4
1.1	Типовая система	4
1.2	Статическая и динамическая типизации	5
1.3	Примитивные типы	6
1.3.1	Определение	6
1.3.2	Теоретический контекст	6
1.3.3	Основные характеристики	6
1.3.4	Примеры в высокоуровневых языках – NEED TO BE FIXED . . .	7
1.3.5	Роль примитивных типов в управляемых средах исполнения . . .	7
2	Постановка задачи	7
3	Обзор существующих решений	8
4	Исследование и построение решения задачи	9
5	Описание практической части	10
6	Заключение	11

1 Введение

Типовая система языка программирования служит фундаментом для формального определения его спецификации, обеспечивает статический анализ программ, поддержку оптимизаций кода и способствует улучшению его структурированности и читаемости. Высокоуровневые языки программирования (Java, C#, Kotlin) предоставляют разработчику мощные абстракции и безопасность кода посредством автоматического управления памятью и объектно-ориентированных парадигм. Многие из этих языков сохранили в себе исторически сложившееся разделение типов на "примитивные" типы и "объектные" или "ссылочные", призванное потенциально улучшить производительность в некоторых случаях. Тем не менее, в то же время оно приводит к несогласованности в системе типов и усложняет достижение полностью единообразной и интуитивной модели программирования. Примитивные типы часто лишены полезных свойств объектов, таких как наследование, полиморфизм, вызов методов без явного упаковывания/распаковывания и использование в качестве аргумента для шаблонных типов.

В данной работе будут теоретически рассмотрены последствия устранения явной разницы между примитивными и объектными типами из высокоуровневого управляемого языка программирования и описаны этапы и результаты реализации.

1.1 Типовая система

Типовая система — это формальная синтаксическая и семантическая структура, сопоставляющая типы и программные конструкции (выражения, переменные, функции) для обеспечения чётко определённого вычислительного поведения. Её основные теоретические и практические назначения:

1) Корректность поведения программы

С введением типовой системы становится возможной статическая верификация соблюдения ограничений типов, не допускающая к выполнению неверно типизированные программы. Данный механизм гарантирует:

- *Безопасность выполнения* — предотвращение неопределённого поведения за счёт блокировки запрещённых операций (например, некорректные обращения к памяти в управляемых средах типа JVM/CLR)
- *Теоретическую обоснованность* — соответствие двум фундаментальным принципам типобезопасности по Райту-Феллейзену:
 1. **Прогресс:** корректно типизированная программа не может застрять в промежуточном состоянии
 2. **Сохранение:** типы остаются согласованными на всех этапах вычисления

2) Уровень абстракции кода

Типовая система обеспечивает инкапсуляцию доменных инвариантов с помощью абстракций типов (абстрактные типы данных и интерфейсы) и позволяет формально специфицировать границы модулей.

3) Возможность оптимизаций

Наличие статической типизации позволяет применять оптимизации на этапе компиляции (например, специализация обобщённых типов и статическое разрешение методов) и минимизировать время исполнения в управляемых средах за счёт уменьшения количества динамических проверок типов.

1.2 Статическая и динамическая типизации

1. Статическая типизация (Java, C#, Kotlin):

- Валидность типа доказывается на этапе компиляции с помощью формальных суждений о принадлежности выражения к типу в данном контексте ($\Gamma \vdash e : \tau$)
- Гарантирует типобезопасность по Райту-Феллейзену
- Критично для управляемых языков программирования из-за возможности проверок безопасности памяти (например, верификатор байт-кода JVM) и эффективности Just-In-Time компиляции за счёт встраивания методов, ориентированных на конкретный тип

2. Динамическая типизация (Python, JavaScript):

- Типы функций и выражений определяются во время исполнения
- Нет формальной гарантии типобезопасности, но есть возможность менять поведение объектов, модулей и классов во время исполнения

Современные управляемые языки высокого уровня (Java, C#) преимущественно основаны на статической типизации, расширенной возможностями динамического анализа во время выполнения. Данный компромиссный подход позволяет сочетать строгость формальной верификации с практической гибкостью разработки, но противоречие между статическими гарантиями и динамической адаптируемостью продолжает оставаться предметом активных исследований.

1.3 Примитивные типы

1.3.1 Определение

Примитивные типы (далее – примитивные типы или примитивы) являются наиболее фундаментальным типом данных в языках программирования. В отличие от объектов, они:

- **Не ссылают на память**, а непосредственно являются значениями, обычно находящимися на стеке. У них отсутствуют:
 - заголовок объекта (object header)
 - таблица виртуальных методов (vtable)
 - ссылочная идентичность (identity)
- Имеют **фиксированную семантику**, определяющуюся стандартами языка (например, Java JLS §4.2, C# ECMA-334 §8.3)
- **Не могут быть подтипами** или наследоваться и являются атомарными (базовыми) элементами в иерархии типов

1.3.2 Теоретический контекст

В теории типов примитивы являются простейшими случаями конструирования типа и не разлагаются на более простые типы. Конструирование сложных типов происходит на основе:

- **Произведения** (кортежи, записи) – `(Int, Bool)`
- **Суммы** (типы-суммы, перечисления) – `data Maybe a = Nothing | Just a`
- **Функции** – `Int → Bool`
- **Рекурсивные типы** – `data List a = Nil | Cons a (List a)`

1.3.3 Основные характеристики

- **Представление в памяти**

Аллокация на стеке* (default for local variables in methods)

- **Производительность на уровне процессора**

1. Арифметические операции с примитивами компилируются непосредственно в нативные инструкции:

```
1      ; x86 assembly for 'a + b'
2      mov eax, [a]
3      add eax, [b]
4
```

2. Локализация в кэше

Линейная модель доступа к памяти (критично для больших вычислений)

1.3.4 Примеры в высокоуровневых языках – NEED TO BE FIXED

Таблица 1: Примитивные типы в различных языках программирования

Тип	Java	C#
Целые числа	int, long	int, long
Числа с плавающей	float, double	float, double
Булевы значения	boolean	bool
Символ	char	char

1.3.5 Роль примитивных типов в управляемых средах исполнения

- **Специальные байт-код инструкции:**
 - В JVM: `iload`, `iadd`, `fstore` для операций с `int`, `float` и др.
 - В CLR: аналогичные IL-команды (`ldc.i4`, `add`)
- **Оптимизации времени исполнения:**
 - Escape-анализ: Размещение временных переменных на стеке вместо кучи
 - Интринсики: Замена вызовов методов (например, `Math.sin()`) на нативные CPU-инструкции

2 Постановка задачи

Необходимо формально изложить суть задачи в данной секции, предоставив такие ясные и точные описания, которые позволят в последующем оценить, насколько разработанное решение соответствует поставленной задаче. Текст главы должен следовать структуре технического задания, включая как описание самой задачи, так и набор требований к ее решению.

3 Обзор существующих решений

Здесь надо рассмотреть все существующие решения поставленной задачи, но не просто пересказать, в чем там дело, а оценить степень их соответствия тем ограничениям, которые были сформулированы в постановке задачи.

4 Исследование и построение решения задачи

Требуется разбить большую задачу, описанную в постановке, на более мелкие подзадачи. Процесс декомпозиции следует продолжать до тех пор, пока подзадачи не станут достаточно простыми для решения непосредственно. Это может быть достигнуто, например, путем проведения эксперимента, доказательства теоремы или поиска готового решения.

5 Описание практической части

Если в рамках работы писался какой-то код, здесь должно быть его описание: выбранный язык и библиотеки и мотивы выбора, архитектура, схема функционирования, теоретическая сложность алгоритма, характеристики функционирования (скорость/-память).

6 Заключение

Здесь надо перечислить все результаты, полученные в ходе работы. Из текста должно быть понятно, в какой мере решена поставленная задача.

Список литературы

- [1] *Mott-Smith, H.* The theory of collectors in gaseous discharges / *H. Mott-Smith, I. Langmuir* // *Phys. Rev.* — 1926. — Vol. 28.
- [2] *Морз, Р.* Бесстолкновительный PIC-метод / *Р. Морз* // Вычислительные методы в физике плазмы / Ed. by Б. Олдера, С. Фернбаха, М. Ротенберга. — М.: Мир, 1974.
- [3] *Киселёв, А. А.* Численное моделирование захвата ионов бесстолкновительной плазмы электрическим полем поглощающей сферы / *А. А. Киселёв, Долгонос М. С., Красовский В. Л.* // Девятая ежегодная конференция «Физика плазмы в Солнечной системе». — 2014.